統計数値データの圧縮と
国勢調査データベース・システム

内　藤　三　義

はじめに

本稿ではコンピュータで扱う統計数値データを対象にした，データ圧縮の新しい技法を説明し，併せて，国勢調査データファイルの総合的データベース化の方法を提案する。

筆者は既に，「日経財務データ」，「家計調査年報」などを対象にした，数値データ圧縮法について私見を述べたが[1]，特定の種類の統計データだけでなく，たいていの統計数値に「汎用」的に利用できると思われるデータ圧縮法へと，これまでの圧縮法を「改良」した。この新しい「圧縮技法」を説明するのが，本稿の第１の目的である。

第２に，国勢調査データについてであるが，磁気テープ媒体でも提供されているこのデータをコンピュータで直接的に利用することは，これまでは利用環境の制約などから，研究者の間にあまり普及してはいなかったようである。このデータを利用する機会は，社会科学分野では非常に多いが，購入費用，データベース・システムとする時に必要な機器施設環境，検索システム作成などの諸問題がネックとなり，一般の教育・研究での利用がなかなか進まないのが現状である。そこで，利用したい磁気テープ媒体などのデータがあれば，比較的簡単にそれをパーソナルにも処理できるデータベース・システムとして作成，稼働させるソフトを開発した。このソフト・システムの内容を示すのが，本稿の第２の目的である。

１．統計数値圧縮法

コンピュータで扱うデータが大容量化するにつれ，より効率的なデータ圧縮技法がますます求められているのは周知のことである。加速度的に進行しているコンピュータ関連技術の革新によって，コンピュータの処理速度の向上，記憶装置のコンパクト化と大容量化，通信速度の高速化が進められ，ますます大きな情報をより速く処理できるようになってきているが，それらをより有効に稼働さすためにも，より正確で効率的なデータ圧縮が求められている。

「データ圧縮技法は本来情報に付帯する冗長性を取り除いて，最大限に環境条件に適したデータの格納・伝送を行うための技術」[2]，であり，アルファベット文字を表すための平均コード長を圧縮する「ハフマンのコード化」[3]など多様な技法[4]がすでに開発されている。しかしそれらは文書やプログラムなどファイル一般を対象にしたものであるので，それらだけを利用することでは，統計数値データに対する高い圧縮を実現することはできない。数値の表現方法としては，バイナリー型で整数や浮動小数点数値を表現する方式や，10進数の１桁を0.5バイトで表現するパック化という表現方法あり，これらは10進数１文字・１バイトで表現するキャラクター型の表現からみれば１つの「圧縮」方法であるとも言える。しかし，これらも，数値の性質が一定している場合に，キャラクター型に対しておおよそ50％程度まで圧縮出来る程度で，高率の圧縮を統計数値一般に保証できるものではない。そこで，統計数値（統計書に記載されるような数値）に限定しながら効果的な圧縮法を考えてみよう。

1－1．統計数値の特性

統計書に記載され，磁気テープなどの記憶媒体に保存される数値の特徴としては次の５つのことをあげることができる。

第１には，数値は概ね10進数で表記される正負実数であるが，整数だけでなく小数点以下が表示されている数値もあるということである。磁気テープなどへの記録のされ方は，パック化されているものもあるが，小数点以下の桁数表示がランダムに変化するケースが含まれている統計書もあり，その場合データ形式の汎用性を確保するためにもＥＢＣＤＩＣコードによるピクチャー形式で記録されているものが多い。

第２に10進数等で表示された数値の有効桁数あるいは最大桁数であるが，これは統計書によって一定していないというものの，９桁を最大（マイナス記号を１桁とすると10桁）とするものが非常に多い。日本の場合，総人口が「億」という10進数９桁の値であることを考えてみれば，概ね９桁以下が多いのは想定もつく。但しそれ以上の最大桁数のものも無いわけでない。

第３には，数値は統計書では表の形で記載されることが多く，磁気テープではその表の１行，もしくは１列が１レコードという単位にまとめられることが多い。その場合レコードの意味を示すための RID（Record Identifire）がレコードの先頭に固定書式で記録されることが多い。

第４には，該当数値データがないとか該当数値が計算できない，あるいは極めて小さいか大きいといった場合に，何らかの欠損値表示が統計書では行われ，磁気テープ・データでも特殊な文字列やコードがこれを示すために使われる。この「欠損値」の全データに占める比率は決して低いものではない。

第５は数値の特性というより，データファイルの特性とも言えるが，この種の数値データは，静的で書き直しのされないデータであり（統計の間違いによって，後に「変更」される可能性が無いわけではないが），別の言葉で言えば読みだし専用，もしくは追記型のデータという性格を持っている。一時的な保存のためだけでなく，恒常的に利用するデータベース・ファイルを圧縮しても問題が起きないのはこの特徴の故である。

以上のような諸特徴が統計数値データには考えられるが，これらの中でデータ圧縮のために考えなければならない問題は，(1)正負数値の混在，(2)整数と小数点表示の値の混在，(3)有効最大桁数が概ね10進数９桁であること，(4)かなりの出現頻度ある欠損値の存在，(5)レコードを単位とした数値データの記録（数値が１項目つづでなく，何項目かまとめて記録されている），という５つの問題であり，データベース・システムとして構成するときには，(6)RID 情報をどう効率的に整理するかという問題が追加される。

では，実際の数値はどのような値の分布になっているのだろうか。家計調査年報・昭和63年，昭和60年国勢調査基本集計・京都府，日経財務データ（1988年まで）を例に，その数値の精度を落とすこと無く表現出来るビット列数と，レコードを単位として見た正負値の混在数，小数点表示の数値の割合と，その桁数などについて見てみると表１のようになった。

３つのデータの，有効最大桁数は10進数９桁であり，実際に記録されている数値は29 bit 以下で表現出来るが，15 bit 以下でも表現出来る数値が全体の90％を越えていることが分かる。レコードを単位としてみると，全て正であるもの，整数を含め繰下がり桁数がレコード中の数値で一定であるものがかなりの率を占めることが分かる。

1－2．N式数値圧縮法

そこでデータ圧縮のために，数値の表現方法を次のように整理しよう。

(1)　正負の判定はレコード単位，もしくは個別数値別に情報を与える。

(2)　小数点付の数値（10進数）はそれを小数点以下の有効桁数分繰り上げて整数化し，その繰上げ桁数を判定できる情報を付加する[5]。

表－1　諸統計データで現れる数値

（1）数値を表現するために必要なビット数

	家計調査年報		国勢調査		日経財務	
数値総数	1245619	100.000	810820	100.000	14388465	100.000
欠損値	247917	19.903	180392	22.248	6424285	44.649
0 bit	31661	2.542	-	-	178419	1.240
1 bit	2951	0.237	33893	4.180	159456	1.108
2 bit	5444	0.437	35117	4.331	264012	1.835
3 bit	9012	0.723	37167	4.584	374479	2.603
4 bit	11229	0.901	40252	4.964	472326	3.283
5 bit	15557	1.249	45922	5.664	602439	4.187
6 bit	20773	1.668	53733	6.627	637024	4.427
7 bit	29009	2.329	63348	7.813	703765	4.891
8 bit	48671	3.907	59824	7.378	777182	5.401
9 bit	73786	5.924	64498	7.955	747923	5.198
10 bit	104640	8.401	48334	5.961	692167	4.811
11 bit	135994	10.918	38014	4.688	615712	4.279
12 bit	134748	10.818	34228	4.221	519061	3.607
13 bit	123344	9.902	26112	3.220	403499	2.804
14 bit	95334	7.654	19259	2.375	289394	2.011
15 bit	54739	4.395	12935	1.595	185844	1.292
16 bit	30762	2.470	8107	1.000	111277	0.773
17 bit	20883	1.677	5330	0.657	61603	0.428
18 bit	15359	1.233	2232	0.275	30725	0.214
19 bit	16925	1.359	1146	0.141	32210	0.224
20 bit	7324	0.588	604	0.074	6509	0.045
21 bit	3833	0.308	297	0.037	2349	0.016
22 bit	2344	0.188	76	0.009	575	0.004
23 bit	1701	0.137	-	-	148935	1.035
24 bit	1150	0.092	-	-	-	-
25 bit	164	0.013	-	-	-	-
26 bit	134	0.011	-	-	-	-
27 bit	204	0.016	-	-	-	-
28 bit	26	0.002	-	-	-	-
29 bit	1	0.000	-	-	-	-

（2）レコードを単位としてみた数値の性質

	家計調査年報		国勢調査		日経財務	
レコード総数	5651	100.000	96416	100.000	52705	100.000
正・整数	1211	21.430	82662	85.735	2709	5.140
正・小数点１桁	4	0.071	2353	2.440	-	-
正・小数点２桁	379	6.707	2762	2.865	-	-
正・小数点３桁	58	1.026	-	-	-	-
正・小数点０～１桁	45	0.796	991	1.028	-	-
正・小数点０～３桁	2484	43.957	7648	7.932	-	-
正負混在・整数	-	-	-	-	49996	94.860
混在・小数点１桁	29	0.513	-	-	-	-
混在・小数点０～１桁	391	6.919	-	-	-	-
混在・小数点０～３桁	1050	18.581	-	-	-	-

(3)　整数化された全ての数値はその絶対値を２進数のビット列で表現するが，固定ではなく，可変ビット列で表し，そのビット列数を判定できる情報を付加する，なお，０(ゼロ)以外を表現するビット列の先頭には必ず１が表れるがこの１ビットは省略する。

(4)　欠損値を表現する特殊なコードを設定する。

1－2－1．N式整数の構造

上記のことを実現するような数値の表現方法として，具体的には次のような「N式整数」を規定する。

数値を表すビット列の並びを，［I_1～Ii K_1～Kk M D_1～Dd］とし，その意味を次のように設定する。

I_1～Ii　：欠損値，もしくは数値を表現するビット数（dの値）を示す
K_1～Kk　：10進数換算の繰り上げ桁数を示す（下に説明するレコード単位制御情報RCI-Nを付加しないとき，このビットは通常3ビットとする）
M　：正／負を示す
D_1～Dd　：先頭に［1］bを付加したビット列が正整数化された数値の値である

ここでI_1～Iiのビット列の意味は次のようにする。

11　：欠損値を表す。（KMD）のビット列は与えられない
0xxxx　：d＝［xxxx］b－2
　但［xxxx］b＝0はゼロ
10xxxx　：d＝14＋［xxxx］b
　但［xxxx］＝［1111］の時は更にI情報が4bit付加され，d＝［xxxx］b＋29とする。

1—2—2．N式レコード制御情報〈RCI-N0とRIC-N1の構造〉

さらに，レコードを単位とした正負の混在や繰上げ桁数を示すために，次のような「N式レコード制御情報（RCI-N0）」（1 or 2）＋5ビットを，レコードの先頭に置く場合もあるとする。

(1)　RCI－N0

(1)—1　正負制御情報（1or2）ビット
0　：レコード内の数値は全て正
10　：レコード内の数値は全て負
11　：正負混在のため，0と欠損値以外では正負制御ビットとして各数値データ内に1ビットを確保している。

(1)—2　繰上げ桁数情報　5ビット
0xxxx　：レコード内の繰上げ桁数は全て同じで［xxxx］b桁
1xxxx　：レコード内の繰上げ桁数は全て同じではない。欠損値以外では桁数情報を与えるため［xxxx］bビットを各数値毎に確保している。

(2)　RCI－N1

なお，この情報は，次のような「N式レコード制御変形情報（RCI-N1）」として書くこともあるとする。

00　：レコード内の数値は全て正整数
01　：レコード内の数値は全て正で，繰上げ桁数情報のため2bitを確保している
100：レコード内の数値は全て正で，全て10進数2桁繰上げて入力されている
101：レコード内の数値は全て正で，繰上げ桁数情報のため1bitを確保している
110：レコード内の数値は全て正で，全て10進数1桁繰上げて入力されている
111：RCI-N0の情報がこの3ビットの後に与えられている

1—2—3．N式数値圧縮法表現の具体例

前述のRCI-N1付のN式整数が一般的なN式数値圧縮法であるが，この方法によって欠損値をふくめた9つの数値からなる1レコードの数値データを表現すると，表2のようになる。事例2の場合，9つの数値を表現する1レコード12バイトは，16進数表記すると［98 02 79 11 21 25 89 90 52 F1 60 08］hとなる。

このN式数値圧縮法を利用して，昭和60年国勢調査基本集計・京都府の圧縮効率を見たとこ

表2　ある1レコードの数値の表現

事　例　1		事　例　2	
RCI-N1	111 11 10010	RCI-N1	100
数値	I　K M　D	数値	I　K M　D
欠損値	11··· ·· · ·····	欠損値	11··· ·· · ·····
0	00000 00 0 ·····	0.00	00000 ·· · ·····
1	00001 00 0 ·····	0.01	00001 ·· · ·····
2	00010 00 0 0·····	1.00	00111 ·· · 100100·····
3	00010 00 0 1·····	2.00	01000 ·· · 1001000····
4	00011 00 0 00·····	3.00	01001 ·· · 00101100···
-7	00011 00 1 11·····	4.00	01001 ·· · 10010000···
102.5	01011 01 0 0000000001·	7.00	01010 ·· · 010111100··
1.025	01011 11 0 0000000001·	10.25	01011 ·· · 0000000001·
総ビット	102 bits(13 bytes)	総ビット	93 bits(12 bytes)

ろ，オリジナルデータは810,820の数値が，各10バイトで入力され，計8,108,200のバイトの容量となるが，この圧縮により得られたデータは1,065,453バイトであり13.14％という高圧縮を達成した。N式数値圧縮法は，表現桁数の大きい数値の比率が増えるほど圧縮率が下がり，キャラクター形式の50％程度まで下がって，実質的な圧縮効果をもたなくなることも有り得るが，表現桁数の少ない数値の出現頻度もかなり存在する統計データの場合，おおよそ20％程度への圧縮は実現できる[6]。

2．国勢調査データファイルのデータベース化

1．で提案した「N式数値圧縮法」を利用し，国勢調査データのデータベース化を試みよう。同データファイルには，統計数値以外にも，各レコードの情報をしめすRID（Record IDentifier）が入力されており，このレコード情報をどのようにインデックス化するかが，アクセス効率を確保するのに重要な問題となる。そこでこのRID情報がどのように入力されているか見てみよう（表－2参照）。

同じ国勢調査データでも，提供されているデータファイルの種類により，RID部分はかなり異なっている。従って作成するデータベース・システムも，この相違を受けて個々にシステム構成することもできる。しかしそうすると，管理しなければならないファイルの数が多くなり，さらにデータベース検索ソフトが複雑になるし，検索手順もそれぞれについて異なり，ユーザーには操作しにくいものになる可能性が強い。インデックス・システムがある程度冗長性を持つことになっても，統一したデータベース・システム構成にするのが，開発にも利用にも都合の良いことのように思える。

2－1．データベース・ファイルの構成

そこで以下のような3つのファイルからこのシステムを構成させる。データ検索者は通常，統計の種類，表番号という大きな分類項目から，都道府県，市区町村と順に小さく絞りながら情報を得ていくものという前提に立っており，重層的に構造化されたデータベース・システムとなっている。国勢調査基本集計を例にして内容を説明する。

2－1－1．メインファイル

統計数値データだけがレコードを単位としたN式数値圧縮法に従って入力されている（RCI-N1付）。1レコードのデータがバイトの途中で終わる時は，残りのビットには［0］b

表－3　国勢調査データファイルの RID 情報

基本集計		従業地・通学地		抽出詳細集計		調査区別集計		地域メッシュ統計	
セル数	5	セル数	5	セル数	5	セル数	5	ファイル名	6
ファイル名	5	ファイル名	5	ファイル名	5	ファイル名	5	ブランク	1
統計表番号	5	統計表番号	5	統計表番号	5	統計表番号	5	地域コード	9
都道府県	2	地域１区分	1	都道府県	2	ブランク	2	ブランク	2
市区町村	3	都道府県	2	市区町村	3	都道府県	2	区画区分	1
ブランク	2	市区町村	3	ブランク	2	市区町村	3	ブランク	1
人口集中地区	2	地域２区分	1	人口集中地区	2	ブランク	2	地形図名	18
分類項目１	3	都道府県	2	分類項目１	3	町丁・字	3	市区町村数	2
分類項目２	3	市区町村	3	分類項目２	3	ブランク	1	都道府県１	2
分類項目３	3	ブランク	3	分類項目３	3	調査区番号	7	市区町村１	3
ブランク	7	分類項目１	3	分類項目４	3	人口集中地区	2	府県市区２	5
		分類項目２	3	ブランク	4	都市計画区分	2	府県市区３	5
		ブランク	4			抽出層符号	6	府県市区４	5
						ブランク	2	｜	｜
								府県市区８	5

数字は各項目が入力されているカラム数（バイト数）を示す

が付加されているので，レコードがバイトの途中から始まることはない。１レコードに含まれるデータ項目数（セル数）はこのファイルの中には与えられていないので，次の２つのファイルがないと，データを再現する（圧縮を解除する）ことはできない。なお，１都道府県データは統計の種類，表番号が同じなら基本的には連続して入力されており，その連続１単位がアドレスファイルの１レコードとなっている。

２－１－２．インデックスファイル

オリジナル・データファイルRIDの大部分の情報を保存する。市区町村コード［B04］，分類項目１～３（B05～B07），ダミー分類項目（B08～B11），該当するデータがメインファイルで記録されているバイト数［REC］，の計９つの数値がN式整数（但，正負ビットと桁数制御ビット無し）の記録形式で入力されている。メインファイルのレコード数とこのファイルのレコード数は同じとなる。なお，ダミー分類項目には欠損値を入力している。

２－１－３．アドレスファイル

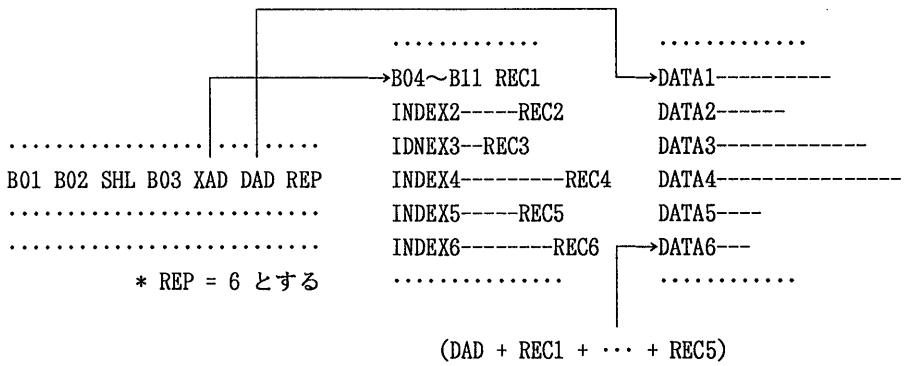
統計のファイル名を適当なコードに書換えた［B01］，表番号［B02］，その表のデータ項目セル［SHL］，都道府県コード［B03］，該当するデータがインデックスファイルやメインファイルに入力されているレコードの総数［REP］，インデックスファイルでのスタートアドレス［XAD］，データファイルのスタートアドレス［DAD］を入力している。このファイルは上記２ファイルの２％程度の容量となるので数値の圧縮は行わず，固定書式で入力されている。

２－１－４．データベースファイルの相互関係

この３つのファイル情報の関係を図示すると図－１のようになる。つまり，B01～B03 の属性を持ったデータはSHL項目の数値データから構成されており，６ケースが入力されているが，DATA1は更にINDEX1に示されたB04～B11という属性を，DATA6はINDEX6 に示される属性をあわせ持つ統計数値である。

このような構造になっているので，ファイル

図－1　データベース・ファイル情報の相互関係



内の情報を書き換えたり，特定のデータを削除しその領域をコンデンスすることなどは非常に難しいが，このシステムはデータベースの作成と利用に関わる２つの長所をもっている。それは情報を連結する（データベースを合併させる）ことが非常に簡単であることと，アドレスファイルをレコードを単位として自由に組み替えることができるということである[7]。

磁気テープから資料－１で示すプログラムなどで変換作成された複数のデータベース・システムの結合は，年度や種類の差異にかかわりなく，同じ操作で実行できる。インデックスとメインファイルに関しては，単純に連結コピーすればよいだけである（連結の順番は同じでなければならない）。アドレスファイルについては，連結コピーの際に前に置かれたインデックスとメインファイルの容量を，後ろに連結した側のアドレスファイルのXADとDAD の値にそれぞれプラスすれば，あとはどのように連結コピーしてもよい。連結の際に何らかの情報操作をする必要があるのは，全体の２％程度の容量にしかならないアドレスファイルだけであり，磁気テープデータ，ファイルを入手の都度，データベース・システムを膨らますことができるという特性をもっている。

またアドレスファイルの各レコードは相互に独立したものなので，レコード単位に並べ替えるのは全く自由であり，また，特定のレコードのみをとりだしたアドレスファイルを作成することも可能である。複数ユーザーに利用させる環境で，ユーザー資格によって利用できる統計を限定したいときなどは，各ユーザーに対応したアドレスデータを抜出して利用させることも出来る。

２－２．データベースの作成プログラム

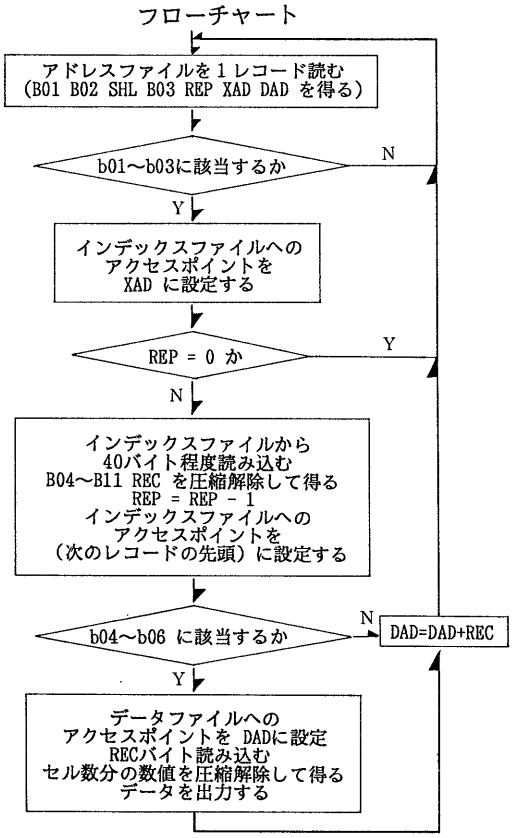
２－１．で示した３つのファイルから構成されるデータベース・ファイルを作成するＰＬ／Ｉ言語によるプログラムは資料－１のようになる。これは国勢調査第１次～第３次基本集計のデータファイルを変換するものであるが，入出力ファイル・変数宣言（11～49）と，一連の初期設定作業（54～63）をした後，入力データファイル１レコードを読み，(68)，分類の１～11（N_BUN(1)～N_BUN(11)），セル数（N_SHELL），整数リスト（N_SUJI(1)～N_SUJI(N_SHELL)），その数値の繰り上がり桁数（N_KETA(1)～N_KETA(N_SHELL)）に値を入れ（69～119），サブプログラムNC_MAINをコールする（121）というものである。サブプログラムには，初期設定NC_INIT，メインルーチンNC_MAIN，最終処理NC_ENDの３つと，そこから更にコールされるもの（NC_COMP4，NC_UCOMP4，NC_STOCK，NC_MCOMP）があるが，すべてあわせても約230行とそれ程大きなものでないので，プログラム内に置いている。入力データファイルの書式と構成により変更する必要があるのは，資料－１で網かけされた部分（およそ70行）だけである。

2－3．データベースへのアクセス法

このデータベースへのアクセスは，検索したい統計種別［b01］，表番号［b02］，都道府県［b03］，市区町村コード［b04］，分類１～分類３［b05～b07］が指定されると，図－２のようになる。

このアクセスと検索を実行するＣ言語による（サブ）プログラムは資料－２に示したが，b01～b03は単一のコード指定，b04～b11はそれぞれにつていコードリストを指定出来るようになっている。ユーザープログラムでの利用の仕方は，(1)最初にデータベースの３つのファイル名と欠損値に割当てる数値を指定してnc_open（84行）をコールする，(2)検索したい条件リストを指定してnc_set（103）をコールする，(3)出力結果を取込む変数のアドレス（ポインタ）を指定してncds（117）をコールするという３つの手順となる。同一条件に該当するデータをすべて取りだしたいなら，(3)を繰返し，別の条件でデータを取りだしたいなら(2)を再設定して(3)を繰返すという処理になる。

図－２　データベース・アクセスのフローチャート



2－4．その他の統計資料のデータベース化

このシステムは，国勢調査データを対象にして設計したものであるが，結果として，書き換え型ではないたいていの統計資料をこのデータベース・システムに組み込むことを可能とする構成になっている。数値データがある属性をもったレコードのような単位で取り扱うことができるもので，その属性を分類する項目を11項目以下にまとめることができるもの（さらに条件をつけるなら，大分類項目３種類以下で一定データ絞り込みが出来，中分類項目を８種類以下に整理できるもの）ならば，このデータベース・システムに組み込むことが可能である。特別に整理しなければならないのは分類項目１～11のコード対応表だけであり，後は，同一のソフト，同一の手順でデータを検索・出力さすことができる。かなり広範囲に利用できる効率的な統計数値データベース・システムであると確信する。

付　記

このシステム構成に基づく国勢調査データベース・システム（BUDAS-POCE:Bukkyo University DAta base System for POpulation CEnsus of japan）が完成している。出力形式は固定書式とLotus-1.2.3用Ｋ３フォーマットのみで，コメント文と都道府県・市区町村名以外は数字のみの出力である。統計表の種類は大学機関として購入したデータファイル量に制限されているが，京都府分（1975,80,85度分）が91年６月より学内の研究・教育用に利用できるようになる予定である。

このシステムの利用の中でユーザー・インターフェースの改良や，出力形式の改良（日本語ラベル付けなど）が求められるが，それらはプリ・ポストプロセッサソフトを適当に組み込

めばよいことであると理解しており，徐々に実行していくつもりである。

なお，このデータベース・システムの作成にあたって，立命館大学計算機センターには多くの便宜をはかって頂いた。深く感謝したい。

注

1) 排稿「大容量データベースの圧縮利用法――日経財務データの圧縮とパソコンでの利用」，『社会学部論叢』第23号，佛教大学学会，1989年12月。「家計調査年報のデータベース化」，同上，第24号，1990年12月。

2) ギルバート・ヘルド（渡部・堀池訳），『データ圧縮技法入門』，啓学出版，1985年「訳者まえがき」より。

3) 例えば英語で書かれた文書の中に実際に表れる文字について考えてみると，出現頻度の高い文字とそうでないものがある。普通英字１文字は１バイト（８ビット）で表されるが，出現頻度の高いものは８ビットより短いコードで，低いものは場合によってはそれより多いビット例のコードで表現しても文書全体としてみれば圧縮出来るというものである。

4) パソコン通信の中で広まっている吉崎栄泰氏が作成したＬＨＡＲＣ法はデータの性質を問わない高圧縮を実現するものである。これは適応的ハフマン法を採用しており，圧縮の解除（解凍）はファイルをシーケンシャルにアクセスすることで行う。

5) コンピュータが扱う数値の記録形式は，２進数が都合の良いことは周知の事であるが，統計表（あるいは統計データファイル）は，それがどのように計算された数値であるにせよ，10進数で表記・記録されるのが一般的である。この数値が整数の場合は問題ないが，小数点以下表示がある場合は直接２進数で表記するとかなり長大なビット列が必要となることがあり，10進数の小数点以下を繰上げた方が，簡略に表現出来ることが多い。

6) Ｎ式数値圧縮法はさらに桁数インデックスに使用するビット数とそのコード化について「改良」することができる。すなわち表－１(1)を見ても各統計書によって，表現必要ビット数の分布は欠損値を別とすると，８ビット，もしくは10ビットを頂点とするような分布となっているので，表現必要ビット数を現すコードをハフマンの記号化法に従って$-LOG_2(P)$（Pはその表現必要ビット数の数値出現頻度）ビットに設定することも出来る。また出現頻度が１％を越えるような数値（１～９など）を，この記号化に直接組み込み，データ表現のためのビットを省略することもできる。このような「改良」の可能性を認識しながら，それを採用しないのは圧縮のためのコード化を，データによって変動させることが，データベースの追記性の障害となることを避けようとしたからである。データベース・システムの運用効率を高めるためのある程度の冗長性は避けられないものと考えたからである。

7) このような「長所」は結果的なものでなく，最初の設計で意図したことでもある。この調査のデータファイルは，集計が完了してから順に公表されるものであり，また我々がデータを入手するのは，必要に応じた地域の統計からである。どのような集め方をしても，結果として同じ手順で使用できるデータ・ベースシステムにする必要があったわけである。

資料１　国勢調査データーベース・システム作成プログラム

```
BUDASP : PROC OPTIONS(MAIN) ;
/* ******************************************************
   *                    MAKE BUDAS-POCE                 *
   *                                                    *
   *          国勢調査・基本集計データファイルの圧縮    *
   *                                                    *
   *             1990-10-24 (REV-91-02-20) 内藤三義     *
   ******************************************************

   ファイル宣言                                          */
   DCL INF FILE RECORD INPUT  ;
   DCL OUF FILE RECORD OUTPUT
               ENV(VB RECSIZE(1000) BLKSIZE(4000)) ;
   DCL IDX FILE RECORD OUTPUT
               ENV(VB RECSIZE(48)   BLKSIZE(2400)) ;
   DCL ADR FILE RECORD OUTPUT
               ENV(FB RECSIZE(24)   BLKSIZE(2400)) ;

  /* ================================================
     変数宣言 ------------------------------------- */
   DCL 1 IDATA,
         2 ID        CHAR(32000) VARYING ;
   DCL 1 TDATA,
       2 SHELL     FIXED BIN(15), 2 NUM      FIXED BIN(30),
       2 P         FIXED BIN(8),  2 TENKEN   CHAR(5),
       2 SUJI1     FIXED(15),     2 SUJI2    FIXED(15),
       2 C10       CHAR(10),      2 C5       CHAR(5),
       2 C3        CHAR(3),       2 C2       CHAR(2) ;
   DCL 1 NCDATA,
       2 NCID,
        3 N_BUN(12) FLOAT BIN(53), 3 N_SHELL FLOAT BIN(53),
        3 N_VAR(2000) FLOAT BIN(53),
        3 N_KET(2000) FIXED BIN(8),
       2 N_MIS     FLOAT BIN(53) INIT(-999999999),
       2 N_OD      CHAR(2048) VARYING,
       2 NCOA,
        3 N_OABUN1 CHAR(2),        3 N_OABUN2 CHAR(4),
        3 N_OASHEL CHAR(2),        3 N_OABUN3 CHAR(4),
        3 N_OAXADR CHAR(4),        3 N_OAXREP CHAR(4),
        3 N_OADADR CHAR(4),
       2 NCPARM,
        3 N_ORGB(4) FLOAT BIN(53), 3 N_NUM    FIXED BIN(30),
        3 N_KER     BIN(4),        3 N_KEP    FIXED BIN(8),
        3 N_PMP     BIN(1),        3 N_REC    FIXED BIN(15),
        3 N_XADR    FLOAT BIN(53), 3 N_XREP   FIXED BIN(31),
        3 N_DADR    FLOAT BIN(53), 3 N_XMAX   FIXED BIN(15),
        3 N_DMAX    FIXED BIN(15),
        3 N_C4      CHAR(4),       3 N_BP     FIXED BIN(15),
        3 N_SUJI    FLOAT BIN(53), 3 N_BS     BIT(64) ;

  /* ================================================
     初期設定とファイル・オープン ------------------- */

     CALL NC_INIT ;
     OPEN FILE (INF), FILE (OUF), FILE (IDX), FILE (ADR) ;
     ON ENDFILE (INF) GOTO FINISH ;
     PUT EDIT ('INPUT FILE NAME OF CENSUS ( 5 CHARA ) --> ')
              (SKIP,A(42)) ;
     GET EDIT (TENKEN) (A(5)) ;
     PUT EDIT ('INPUT SENSUS NO. ( UNDER 65535 ) -------> ')
              (SKIP,A(42)) ;
     GET LIST (SUJI1) ;
       N_BUN(1) = SUJI1 ;
     N_BUN(9) = N_MIS; N_BUN(10) = N_MIS; N_BUN(11) = N_MIS ;
  /* ================================================
     読み込み開始 ----------------------------------- */

  START : READ FILE (INF) INTO (ID) ;
     NUM = NUM + 1;
     OSIZE = OSIZE + LENGTH(ID) ;
     IF SUBSTR(ID,6,5) = TENKEN  THEN GOTO STEP1;
     PUT EDIT ('DATA ERRROR  CASE NO.--> ',NUM)
              (SKIP,A(25),F(7)) ;
     GOTO START ;

  /* アドレスデータ (分類2, 3, シェル) の取込み ------- */
     STEP1 :
       C5 = SUBSTR(ID,1,5) ;  IF C5='     ' THEN C5 = '    0' ;
         N_SHELL = C5 ;    SHELL = C5 ;
       C3 = SUBSTR(ID,11,3) ; IF C3 = '   ' THEN C3 = '  0' ;
         SUJI1 = C3 ;      SUJI1 = SUJI1 * 100 ;
       C2 = SUBSTR(ID,14,2) ; IF C2 = '  ' THEN C2 = ' 0' ;
         SUJI2 = C2 ;      N_BUN(2) = SUJI1 + SUJI2 ;
       C2 = SUBSTR(ID,16,2) ; IF C2 = '  ' THEN C2 = ' 0' ;
         N_BUN(3) = C2 ;

  /* インデックス (分類4～8) の取込み ---------------- */
     STEP2 :
       C3 = SUBSTR(ID,18,3) ; IF C3 = '   ' THEN C3 = '  0' ;
         N_BUN(4) = C3 ;
       C2 = SUBSTR(ID,23,2) ;
       IF C2 = '00' THEN  N_BUN(5) = 1 ;
       ELSE               N_BUN(5) = 0 ;
       DO I = 1 TO 3 ;
         C3 = SUBSTR(ID,I*3+22,3) ;
         IF C3 = '   ' THEN N_BUN(I + 5) = N_MIS ;
         ELSE               N_BUN(I + 5) = C3 ;
       END ;

  /* 実データの取込み ------------------------------- */
     STEP3 :
       DO I = 1 TO SHELL ;
         C10 = SUBSTR(ID,I*10+31,10) ;
         IF C10 = '          ' THEN GOTO STE31 ;
         IF C10 = '         ...' THEN GOTO STE31 ;
         IF C10 = '          ' THEN GOTO STE31 ;
         DO WHILE (SUBSTR(C10,10,1) = ' ') ;
           C10 = ' ' !! SUBSTR(C10,1,9) ;
         END ;
         P= INDEX(C10,'.') ;
         IF P = 0 THEN  N_KET(I) = 0 ;
         ELSE DO ;
           N_KET(I) = 10 - P ;
           SUBSTR(C10,1,P) = ' ' !! SUBSTR(C10,1,P-1) ;
         END ;
         N_VAR(I) = C10 ;
         GOTO STE32 ;
         STE31 : N_VAR(I) = N_MIS ;   N_KET(I) = 0;
       STE32 : END ;


       CALL NC_MAIN ;

    GOTO START ;

    /* ================================================
            SUB ROUTINE
       ================================================ */

    /* 初期化 ---------------------------------------- */
    NC_INIT : PROC ;
       N_XADR = 0 ; N_DADR = 0 ; N_XREP = 0 ;
       N_NUM  = 0 ; N_DMAX = 0 ; N_XMAX = 0 ; N_BUN  = N_MIS ;
    END NC_INIT ;

    /* メイン・サブルーチン -------------------------- */
    NC_MAIN : PROC ;
      DCL ( NC_P, NC_M ) FIXED BIN(15),
          NC_BT6  BIT(6),  NC_BT4  BIT(4) ;

    /* ···· 異なる表などの時アドレスの最終計算と書きだし ··· */
       N_NUM = N_NUM + 1 ;
       IF N_NUM = 1  THEN  GOTO NC_STEP1 ;

       IF N_BUN(1) = N_ORGB(1) & N_BUN(2) = N_ORGB(2)
                   & N_BUN(3) = N_ORGB(3) THEN GOTO NC_STEP2 ;
       N_SUJI = N_XREP ;   CALL  NC_UCOMP4 ; N_OAXREP = N_C4 ;
       N_XREP = 0 ;
       WRITE FILE (ADR) FROM(NCOA) ;

    /* ···· 異なる表などの時当たらしアドレスの計算 ····· */
      NC_STEP1 : N_ORGB(1) = N_BUN(1) ;  N_ORGB(2) = N_BUN(2) ;
                 N_ORGB(3) = N_BUN(3) ;  N_ORGB(4) = N_SHELL ;
       N_SUJI = N_BUN(1) ; CALL NC_UCOMP4 ;
                           N_OABUN1 = SUBSTR(N_C4,1,2) ;
       N_SUJI = N_BUN(2) ; CALL NC_UCOMP4 ;  N_OABUN2 = N_C4 ;
       N_SUJI = N_SHELL ;  CALL NC_UCOMP4 ;
                           N_OASHEL = SUBSTR(N_C4,1,2) ;
       N_SUJI = N_BUN(3) ; CALL NC_UCOMP4 ;  N_OABUN3 = N_C4 ;
       N_SUJI = N_XADR ;   CALL NC_UCOMP4 ;  N_OAXADR = N_C4 ;
       N_SUJI = N_DADR ;   CALL NC_UCOMP4 ;  N_OADADR = N_C4 ;

    /* ···· 同一表でセル数が異なれば WARNING ·········· */
      NC_STEP2 :  N_XREP = N_XREP + 1 ;
        IF N_SHELL ^= N_ORGB(4) THEN  PUT EDIT
        ('*WARNING* SHELL DATA ERROR CASE NO.   TRUE ->  REAL',
                              [',N_NUM,'] [',N_ORGB(4),
         ']->[',N_SHELL,']')
        (SKIP,A(51),SKIP,A(27),F(7),A(3),F(5),A(4),F(5),A(1)) ;

    /* ···· 桁数インデックスのためのパラミータの取得 ····· */
      NC_STEP3 :  N_KER = -1 ;  N_KEP = -1 ;  N_PMP = 0;
                  NC_P  = 0 ;   NC_M = 0 ;
        DO I = 1 TO N_SHELL ;
          IF N_VAR(I)  ^= N_MIS THEN DO ;
            IF N_KER  ^= N_KET(I)  THEN  N_KEP = N_KEP + 1 ;
            IF N_KER  <  N_KET(I)  THEN  N_KER = N_KET(I) ;
            IF N_VAR(I) > 0        THEN  NC_P  = NC_P + 1 ;
            IF N_VAR(I) < 0        THEN  NC_M  = NC_M + 1 ;
          END ;
        END ;
        IF N_KEP = -1  THEN  N_KEP = 0 ;
        IF N_KER = -1  THEN  N_KER = 0 ;

    /* ···· 桁数インデックス RCI-NO の決定 ··········· */
      NC_STEP4 :  N_REC = 0 ;  N_OD = '' ;   N_BP  = 1 ;
        SUBSTR(N_BS,1,3) = '111' ;     N_BP = 4 ;
        IF NC_M = 0 THEN DO ;
          SUBSTR(N_BS,N_BP,1) = '0' ;  N_BP=N_BP+1 ;  N_PMP=0 ;
```

```
189       END ;
190       ELSE IF NC_P = 0 THEN DO ;
191         SUBSTR(N_BS,N_BP,2) = '10' ; N_BP=N_BP+2 ;  N_PMP=0 ;
192       END ;
193       ELSE DO ;
194         SUBSTR(N_BS,N_BP,2) = '11' ; N_BP=N_BP+2 ;  N_PMP=1 ;
195       END ;
196       NC_BT4 = N_KER ;
197       IF N_KEP  THEN DO ;
198         SUBSTR(N_BS,N_BP,1) = '1' ;
199         N_KER = 5 - INDEX(NC_BT4,'1') ;  NC_BT4 = N_KER ;
200       END ;
201       ELSE SUBSTR(N_BS,N_BP,1) = '0' ;
202       N_BP = N_BP + 1 ;
203       SUBSTR(N_BS,N_BP,4) = NC_BT4 ;      N_BP  = N_BP + 4 ;
204
205   /* ···· 桁数インデックスを RCI-N1 への変形 ·····  */
206     NC_STEP5 :
207       NC_BT6 = SUBSTR(N_BS,4,6) ;
208       IF NC_BT6 = '000000' THEN DO ;
209         SUBSTR(N_BS,1,2) = '00' ;        N_BP = 3 ;
210       END ;
211       IF NC_BT6 = '010010' THEN DO ;
212         SUBSTR(N_BS,1,2) = '01' ;        N_BP = 3 ;
213       END ;
214       IF NC_BT6 = '000010' THEN DO ;
215         SUBSTR(N_BS,1,3) = '100' ;       N_BP = 4 ;
216       END ;
217       IF NC_BT6 = '010001' THEN DO ;
218         SUBSTR(N_BS,1,3) = '101' ;       N_BP = 4 ;
219       END ;
220       IF NC_BT6 = '000001' THEN DO ;
221         SUBSTR(N_BS,1,3) = '110' ;       N_BP = 4 ;
222       END ;
223
224   /* ···· 実データ部分の圧縮書きだし ····· */
225    NC_STEP6 :
226      DO I = 1 TO N_SHELL ;
227        N_KER  = N_KET(I) ;
228        N_SUJI = N_VAR(I) ;  CALL NC_MCOMP ;  CALL NC_STOCK ;
229      END ;
230      IF N_BP > 1 THEN DO ;
231        SUBSTR(N_BS,N_BP,7) = '0000000' ;  N_BP = 9 ;
232        CALL NC_STOCK ;
233      END ;
234      IF N_DMAX < N_REC  THEN  N_DMAX = N_REC ;
235      N_DADR = N_DADR + N_REC ;
236      WRITE FILE (OUF) FROM (N_OD) ;
237
238   /* ···· インデックスデータの最終計算と書きだし ····· */
239    NC_STEP7 : N_KEP = 0 ;   N_PMP    = 0 ;      N_OD  = '' ;
240               N_BP  = 1 ;   N_BUN(12) = N_REC ;  N_REC = 0 ;
241      DO I = 4 TO 12 ;
242        N_SUJI = N_BUN(I) ;  CALL NC_MCOMP ;   CALL NC_STOCK ;
243      END ;
244      IF N_BP > 1 THEN DO ;
245        SUBSTR(N_BS,N_BP,7) = '0000000' ;  N_BP = 9 ;
246        CALL NC_STOCK ;
247      END ;
248      IF N_XMAX < N_REC  THEN  N_XMAX = N_REC ;
249      N_XADR = N_XADR + N_REC ;
250      WRITE FILE (IDX) FROM (N_OD) ;
251   END NC_MAIN ;
252
253   /* 最終アドレスデータの書きだし ------------------------- */
254   NC_END : PROC ;
255      N_SUJI = N_XREP ;   CALL NC_UCOMP4 ;   N_OAXREP = N_C4 ;
256      WRITE FILE (ADR) FROM (NCOA) ;
257      N_SUJI = N_XADR ;   CALL NC_UCOMP4 ;   N_OAXADR = N_C4 ;
258      N_SUJI = N_DADR ;   CALL NC_UCOMP4 ;   N_OADADR = N_C4 ;
259      UNSPEC(N_OABUN1) = 'FFFF'B4 ;
260      UNSPEC(N_OABUN2) = 'FFFFFFFF'B4 ;
261      UNSPEC(N_OASHEL) = 'FFFF'B4 ;
262      UNSPEC(N_OABUN3) = 'FFFFFFFF'B4 ;
263      UNSPEC(N_OAXREP) = 'FFFFFFFF'B4 ;
264      WRITE FILE (ADR) FROM (NCOA) ;
265
266      PUT EDIT ('       TOTAL RECORD NUMBER=',N_NUM)
267               (SKIP,A(26),F(15)) ;
268      PUT EDIT ('          FILE SIZE OF DATA=',N_DADR)
269               (SKIP,A(26),F(15)) ;
270      PUT EDIT ('                    INDEX=',N_XADR)
271               (SKIP,A(26),F(15)) ;
272      PUT EDIT ('  MAX RECROD SIZE OF DATA=',N_DMAX)
273               (SKIP,A(26),F(15)) ;
274      PUT EDIT ('                    INDEX=',N_XMAX)
275               (SKIP,A(26),F(15)) ;
276   END NC_END ;
277
278   /* ------------------------------------------------
279            SUB-SUB ROUTINE
280      ------------------------------------------------ */
281
282   /* ····  GET LONG (4 BYTES) ·· THIS PROCESS IS NOT USED  */
283   NC_COMP4 : PROC ;
284     DCL  NC_BT32  BIT(32),  NC_BN31 FIXED BIN(31) ;
285     NC_BN31 = N_SUJI ;   NC_BT32 = UNSPEC(NC_BN31) ;
286     UNSPEC(HC4)= SUBSTR(NC_BT32,25,8) !! SUBSTR(NC_BT32,17,8)
287                !! SUBSTR(NC_BT32,9,8) !! SUBSTR(NC_BT32,1,8) ;
288   END NC_COMP4 ;
289
290   /* ···· GET UNSIGNED LONG (4 BYTES) ····· */
291   NC_UCOMP4 : PROC ;
292     DCL  NC_BT32  BIT(32),  NC_BT64  BIT(64),
293          NC_BN7   FIXED BIN(7) ;
294       NC_BT64   = UNSPEC(N_SUJI) ;
295       NC_BN7 = SUBSTR(NC_BT64,2,7) ;
296       NC_BT32   = '00000000'B4 ;
297       IF NC_BN7 = 0   THEN     UNSPEC(N_C4) = NC_BT32 ;
298       ELSE DO ;
299         NC_BN7  = (NC_BN7 - 64) * 4 ;
300         SUBSTR(NC_BT32,33 - NC_BN7,NC_BN7) =
301                                  SUBSTR(NC_BT64,9,NC_BN7) ;
302         UNSPEC(N_C4) = SUBSTR(NC_BT32,25,8)
303              !! SUBSTR(NC_BT32,17,8) !! SUBSTR(NC_BT32, 9,8)
304              !! SUBSTR(NC_BT32, 1,8) ;
305       END ;
306   END NC_UCOMP4 ;
307
308   /* ···· STOCK ONE CHARACTER IN [N_OD] ····· */
309   NC_STOCK : PROC ;
310     DCL NC_C1  CHAR(1),  NC_BT8  BIT(8) ;
311       DO WHILE ( N_BP > 8 ) ;
312         NC_BT8 = SUBSTR(N_BS,1,8) ;  UNSPEC(NC_C1) = NC_BT8 ;
313         N_BS   = SUBSTR(N_BS,9,56) ; N_BP  = N_BP - 8 ;
314         N_OD   = N_OD !! NC_C1 ;     N_REC = N_REC + 1 ;
315       END ;
316   END NC_STOCK ;
317
318   /* ····· MAKE ONE COMPRESS DATA ON N_BS ····· */
319   NC_MCOMP : PROC ;
320     DCL NC_BT64  BIT(64),  NC_BT56  BIT(56),  NC_BT7  BIT(7),
321         NC_BT4   BIT(4),   NC_BT1   BIT(1),   NC_P    BIN(8),
322         NC_BN7   FIXED BIN(7) ;
323     IF N_SUJI ^= N_MIS  THEN  GOTO NC_MSTE1 ;
324       SUBSTR(N_BS,N_BP,2) = '11' ;      N_BP   = N_BP + 2 ;
325       GOTO NC_MSTE2 ;
326     NC_MSTE1 :  NC_BT64 = UNSPEC(N_SUJI) ;
327       NC_BT1  = SUBSTR(NC_BT64,1,1) ;
328       NC_BN7  = SUBSTR(NC_BT64,2,7) ;
329       IF NC_BN7 ^= 0  THEN DO ;
330         NC_BN7  = (NC_BN7 - 64) * 4 ;
331         NC_BT56 = SUBSTR(NC_BT64,9,56) ;
332         NC_P    = INDEX(NC_BT56,'1') ;
333         NC_BN7  = NC_BN7 - NC_P + 1 ;
334       END ;
335       NC_BT7  = NC_BN7 ;
336       IF NC_BN7 < 16 THEN DO ;
337         SUBSTR(N_BS,N_BP,5) = SUBSTR(NC_BT7,3,5) ;
338         N_BP = N_BP + 5 ;
339       END ;
340       ELSE DO ;
341         SUBSTR(N_BS,N_BP,2) = '10' ;   N_BP = N_BP + 2 ;
342         SUBSTR(N_BS,N_BP,4) = SUBSTR(NC_BT7,4,4) ;
343         N_BP = N_BP + 4 ;
344       END ;
345       IF N_KEP THEN DO ;
346         NC_BT4 = N_KER ;
347         SUBSTR(N_BS,N_BP,N_KEP) =
348                                  SUBSTR(NC_BT4,5 - N_KEP,N_KEP) ;
349         N_BP = N_BP + N_KEP ;
350       END ;
351       IF NC_BN7 = 0  THEN  GOTO NC_MSTE2 ;
352       IF N_PMP THEN DO ;
353         SUBSTR(N_BS,N_BP,1) = NC_BT1 ;   N_BP  = N_BP + 1 ;
354       END ;
355       IF NC_BN7 = 1  THEN  GOTO NC_MSTE2 ;
356       NC_BN7 = NC_BN7 - 1 ;
357       SUBSTR(N_BS,N_BP,NC_BN7) =
358                                  SUBSTR(NC_BT56,NC_P + 1,NC_BN7) ;
359       N_BP = N_BP + NC_BN7 ;
360   NC_MSTE2 : END NC_MCOMP ;
361
362   /* ========== END SUBROUTINE ========================== */
363
364   FINISH :
365       PUT EDIT ('END PROCESS ORIGINAL SIZE=',OSIZE)
366                (SKIP,A(26),F(15)) ;
367       CALL NC_END ;
368       CLOSE FILE (INF), FILE (OUF), FILE (IDX), FILE (ADR) ;
369   END BUDASP ;
```

資料2　N式社会統計データーベース検索出力プログラム

```
/* ======================================================

    Naito's Social Statistics Compressed Database System

        N式社会統計圧縮データベース・システム

            ncds.c  v1.0    1991-01-15 内藤 三義
    ====================================================== */

#include <stdio.h>
#include <conio.h>      /* kbhit    getch   */
#include <process.h>    /* exit             */

/*  +-------------------------------------------------------+
    I nc_open(fln,mv) NCDS ファイルオープンと欠損値の指定  I
    I                       char *(*fln)       double mv  I
    I   ..................................................I
    I   fin     :   アドレスファイル・パス名              I
    I   fin+1   :   インデックスファイル・パス名          I
    I   fin+2   :   データファイル・パス名                I
    I   mv      :   欠損値に割り当てる数値                I
    I   return = 0 : 正常終了                             I
    I          = 1 : アドレスファイルがオープン出来ない   I
    I          = 2 : インデックスファイルがオープン出来ないI
    I          = 4 : データファイルがオープン出来ない     I
    +-------------------------------------------------------+
    I nc_close()    NCDS ファイルのクローズ               I
    I       return値なし                                  I
    +-------------------------------------------------------+
    I nc_set(list,parm) NCDS 検索条件の設定               I
    I                       long *(*list)      int parm   I
    I   ..................................................I
    I   (1) 検索したい分類1～11のコード (リスト) の       I
    I       先頭アドレスを list～list+11 に設定する       I
    I   (2) parm = 0 : 現在のアドレスポイントから検索     I
    I            = 1 : アドレスファイルの最初から検索する I
    I                                                     I
    I   return = 0 : 正常終了                             I
    I          1 : ファイルがオープンされてない           I
    +-------------------------------------------------------+
    I ncds(data,keta,parm)  NCDS 検索と出力               I
    I               double *data    char *keta  int parm  I
    I   ..................................................I
    I   (1) data    :   出力する数値を収めるポインタ      I
    I         data[0]   :   レコード番号                  I
    I         data[1]   :   セル数                        I
    I         data[2～4]  :   分類1～3                    I
    I         data[5～7]  :   xadr, xrep, dadr            I
    I         data[8]   :   相対レコード番号              I
    I         data[9～16] :   分類4～11                   I
    I         data[17～*] :   データ・リスト              I
    I         data[* + 1] :   欠損値                      I
    I   (2) keta    :   出力する数値を収めるポインタ      I
    I         keta[0～* ] :   対応する繰上がり桁数        I
    I         keta[* + 1] :   -1                          I
    I   (3) parm = 1 : アドレス情報のみ出力               I
    I            = 2 : アドレス・インデックス情報を出力   I
    I            = 3 : アドレス・インデックス・データ情報 I
    I                  を出力                             I
    I   ..................................................I
    I   return =  0 : 正常終了                            I
    I          = -1 : アドレスファイル・エラー            I
    I          = -2 : インデックスファイル・エラー        I
    I          = -3 : データファイル・エラー              I
    I          =  1 : 対応データはない(eof of address file)I
    I          =  2 : 検索条件が指定されていない          I
    I          =  3 : parm に 1～3 以外が指定された       I
    +-------------------------------------------------------+  */

struct  {   unsigned int    bun1;   unsigned long   bun2;
            unsigned int    shel1;
            unsigned long   bun3,   xadr,   repeat, dadr;
        }   s_adr;
FILE    *fp_a,      *fp_x,      *fp_d;
double  d_missing,  d_xdat[12], *dp_1,  *dp_2,  *dp_3;
long    l_anum,     l_xnum,     l_adat[9],
                    *lp_list[11],       *lp_1;
int     i_fflag = 0,i_xdp,      i_xd1,  i_dparm = -1,
        i_lparm = -1;
char    c_xd[2200], c_dd[2048],         *cp_1;

/* ====================================================== */
int nc_open( char *(*fln),  double mv )
{   int flag = 0;
    if ((fp_a = fopen(*fln,"rb")) != NULL)          flag++;
    fln++;  if ((fp_x = fopen(*fln,"rb")) != NULL)  flag += 2;
    fln++;  if ((fp_d = fopen(*fln,"rb")) != NULL)  flag += 4;
    d_missing = mv;     i_fflag = flag;
    flag = 7 - flag;
    if ( flag )             nc_close();
    else            {   l_anum = 0;     l_xnum = -1;    }
    return( flag );
}
/* ====================================================== */
nc_close()
{   if ((i_fflag & 1) == 1)     fclose( fp_a );
    if ((i_fflag & 2) == 2)     fclose( fp_x );
    if ((i_fflag & 4) == 4)     fclose( fp_d );
    i_fflag = 0;
}
/* ====================================================== */
int nc_set( long *(*list), int parm )
{   int i,  flag = 0;
    for (i = 0; i < 11; i++)
    {   lp_list[i] = *list; list++; }
    l_lparm = 0;
    if (parm == 1)
    {   if (i_fflag == 7)
        {   rewind( fp_a ); l_anum = 0; l_xnum = -1;
            i_dparm = -1;   }
        else        {   flag = 1;       return( flag ); }
    }
    return ( flag );
}
/* ====================================================== */
int ncds( double *data, char *keta, int parm )
{   int i, j, flag;

    if (i_fflag != 7)
    {   flag = 7 - i_fflag;         return( flag );     }
    if (i_lparm)
    {   flag = 2;                   return( flag );     }
    dp_1 = data;            cp_1 = keta;

    if (parm == 1)
    {   flag = adrcheck();          return( flag );     }

    dp_2 = dp_1;        dp_2 += 8;
    if (parm == 2)
    {   flag = idxcheck();          return( flag );     }

    dp_3 = dp_1;        dp_3 += 17;
    if (parm == 3)
    {   int rec, shel, fsp = 0, op = 0;
        if (i_dparm)
        {   if (flag = idxcheck())  return( flag );     }

        rec = (int)d_xdat[9];       shel = (int)l_adat[1];
        flag = -3;      i_dparm = -1;
        if (fseek(fp_d, l_adat[7], fsp) != 0)
                                    return( flag );
        if (fread(&c_dd[0], rec, 1, fp_d) != 1)
                                    return( flag );
        rcomp( &c_dd[0], dp_3, cp_1, shel, op );
        flag = 0;                   return( flag );
    }
    flag = 3;                       return( flag );
}

/* ---- アドレスファイルのチェック  ---------------------
            return   -1:file error,  1:end of file
   ------------------------------------------------------ */
int adrcheck()
{   int i, flag, rec = 24;
    double  *dp;
    dp = dp_1;      l_xnum = -1;        i_dparm = -1;
    while((fread((char *)&s_adr, rec, 1, fp_a) == 1))
    {   l_anum++;
        if (s_adr.bun1 == 0xffff)
        {   flag = 1;               return( flag ); }
        l_adat[2] = (long)s_adr.bun1;l_adat[3] = s_adr.bun2;
        l_adat[4] = (long)s_adr.bun3;
        flag = 0;
        for (i = 0; i < 3; i++)
        {   if ((*lp_list[i] != -1) &&
                (*lp_list[i] != l_adat[2 + i])) flag++;
        }
        if (flag == 0)
        {   l_adat[0] = l_anum;
            l_adat[1] = (long)s_adr.shel1;
            l_adat[2] = (long)s_adr.bun1;
            l_adat[3] = s_adr.bun2;
            l_adat[4] = (long)s_adr.bun3;
            l_adat[5] = s_adr.xadr;
            l_adat[6] = s_adr.repeat;
            l_adat[7] = s_adr.dadr;
            l_xnum = 0;
            for (i = 0; i < 8;  i++)
            {   *dp = (double)l_adat[i];  dp++; }
            *dp = -1;
            return( flag );
    }   }
    flag = -1;  return( flag );
}

/* ---- インデックス・ファイルのチェック  ----------------
    return   -2:file error,  2:end of one address cluster
   ------------------------------------------------------ */
int idxcheck()
```

```
{   int i, j, flag, fsp = 0,    rec = 2048, cx = 9, op = 1;
    long    temp;
    double  *dp;
    flag = 1;       dp = dp_2;      i_dparm = -1;
    while(flag)
    {   if (l_xnum == -1)
        {   if (flag = adrcheck())      return( flag ); }
        if (l_xnum == 0)
        {   if (fseek(fp_x, l_adat[5], fsp) != 0)
            {   flag = -2;                  return( flag ); }
            fread(&c_xd[0], rec, 1, fp_x);
            i_xdp = 0;  i_xdl = 2048;
        }
        else
        {   l_adat[7] += (long)d_xdat[8];
            if (i_xdp > (i_xdl - 48))
            {   j = 0;
                for (i = i_xdp; i < 2048; i++)
                {   c_xd[j] = c_xd[i];      j++;    }
                i_xdp = 0;      i_xdl = 2048 + j;
                fread(&c_xd[j], rec, 1, fp_x);
        }   }
        l_xnum++;       d_xdat[0] = (double)l_xnum;
        i_xdp +=
            rcomp( &c_xd[i_xdp], &d_xdat[1], cp_l, cx, op );
        for (i = 1; i < 9; i++)
        {   lp_l = lp_list[i + 2];  flag = 0;
            if (*lp_l != -1)
            {   flag = 1;   temp = (long)d_xdat[i];
                while((*lp_l != -1) && (flag))
                {   if (temp == *lp_l)  flag = 0; lp_l++;  }
            }
            if (flag)               i = 9;
        }
        if (l_xnum == l_adat[6])    l_xnum = -1;
    }
    for (i = 0; i < 9; i++) {   *dp = d_xdat[i];  dp++; }
    *dp = -1;   i_dparm = 0;
    return( flag );
}

/* ····· N式数値圧縮法からの復帰 ····· */
int rcomp( char *si, double *di, char *bx, int cx, int i )
{   char    ah,al,bl,bh,cl,ch,dl,dh;
    double  ax;
    int     rec = 0;

/*  ·····
    ·   al : one character              cl : dummy          ·
    ·   ah : keta control bit           ch : keta su        ·
    ·   bl : current attack bit(0-7)    dl : databit length ·
    ·   bh : +- contorol parameter      dh : +- sign        ·
    ·                                                       ·
    ·   di : address of output data(long)                   ·
    ·   si : address of input data(char)                    ·
    ·   bx : address of output data for keta(char)          ·
    ·····
    ------------------------------------------------
            正負 bit、桁数 bitインデックスの把
    ------------------------------------------------ */
    dh = ah = bh = cl = 0;

    if (i == 1) {   ah = bh = dh = dl = ch = 0; bl = 7; }
    else
    {   al = *si;   si++;   rec++;  bl = 1;
        if (al < 0) cl = 2;
        al *= 2;
        if (al < 0) cl++;
        if (cl == 0)        {   ah = bh = dh = ch = 0;  }
        else if (cl == 1)   {   ah = 2; bh = dh = 0;    }
        else
        {   cl *= 2;    bl++;   al *= 2;
            if (al < 0) cl++;
            if (cl == 4)    {   ah = bh = dh = 0; ch = 2; }
            else if (cl == 5) { ah = 1;   bh = dh = 0;    }
            else if (cl == 6) { ah = bh = dh = 0; ch = 1; }
            else cl = -1;
    }   }
    if (cl < 0)
    {   cl = 0; al *= 2;    bl++;
        if (al < 0)
        {   al *= 2;    bl++;
            if (al < 0) bh = 1;
            else    {   bh = 0; dh = 1; }
        }
        else    {   bh = 0; dh = 0; }
        al *= 2;    bl++;
        if (al < 0) ah = 1;
        else        ah = 0;
        cl = 0;
        for (i = 0; i < 4; i++)
        {   bl++;   al *= 2;    cl *= 2;
            if (bl == 8) { al = *si; si++; rec++; bl = 0; }
            if (al < 0) cl++;
        }
        if (ah)     ah = cl;
        else        ch = cl;
    }
    if (ah == 0)
        for (i = 0; i < cx; i++)    {   *bx = ch;   bx++; }

/* ·····
    ·   bh = 1 : 正負制御ビットあり                         ·
    ·     bh = 0 なら ( dh = 0 : 全て正／dh = 1 : 全て負 ) ·
    ·   ah = 桁数制御ビット数 ( = 0 はなし )                ·
    ····· */

    ------------------------------------------------
            実データの圧縮解除
    ------------------------------------------------ */
    while (cx > 0)
    {   cx--;
        dl = 0;
        for (i = 0; i < 2; i++)
        {   bl++;       al *= 2;    dl *= 2;
            if (bl == 8)
            {   al = *si;   si++;   rec++;  bl = 0; }
            if (al < 0)     dl++;
        }
        if (dl == 3)
        {   ax = d_missing; if (ah) { *bx = 0; bx++; } }
        else
        {
            /*      実データ記録 bit 数の取得           */
            cl = 3;     if (dl == 2)    cl = 4;
            for (i = 0; i < cl; i++)
            {   bl++;       al *= 2;    dl *= 2;
                if (bl == 8)
                {   al = *si;   si++;   rec++;  bl = 0; }
                if (al < 0)     dl++;
            }
            if (dl > 16)    dl -= 16;

            /*      小数点以下桁数の取得                */
            if (ah)
            {   ch = 0;
                for (i = 0; i < ah; i++)
                {   bl++;       al *= 2;    ch *= 2;
                    if (bl == 8)
                    {   al = *si;   si++;   rec++;  bl = 0; }
                    if (al < 0)         ch++;
                }
                *bx = ch;   bx++;
            }

            /*      0 or 欠損値以外の実データの取得     */
            ax = 0;
            if (dl)
            {   ax++;
                if (bh)
                {   bl++;       al *= 2;        dh = 0;
                    if (bl == 8)    {   al = *si;   si++;   rec++;  bl = 0; }
                    if (al < 0)     dh = 1;
                }
                dl--;
                if (dl)
                {   for (i = 0; i < dl; i++)
                    {   bl++;       al *= 2;    ax *= 2;
                        if (bl == 8)    {   al = *si; si++; rec++; bl = 0;  }
                        if (al < 0)     ax++;
                    }   }
                if (dh)     ax = -ax;
        }   }
        *di = ax;           di++;
    }
    *di = d_missing;        *bx = 0;
    return( rec );
}
```